Todays' Slides

http://bit.ly/ANLP-Lecture7

Sequence to Sequence

The thing that inspired this course:

https://github.com/alvations/kopitiam/blob/master/Kopitiam - PyTorch RNN Seq2Seq.ipynb

ISO 29990:2010

NATIONAL INFOCOMM COMPETENCY FRAMEWORK

NUS National University of Singapore

ISS INSTITUTE OF SYSTEMS SCIENCE

Text Processing using Machine Learning

Attention

Liling Tan

27 Feb 2019

Overview

Lecture

- Attention

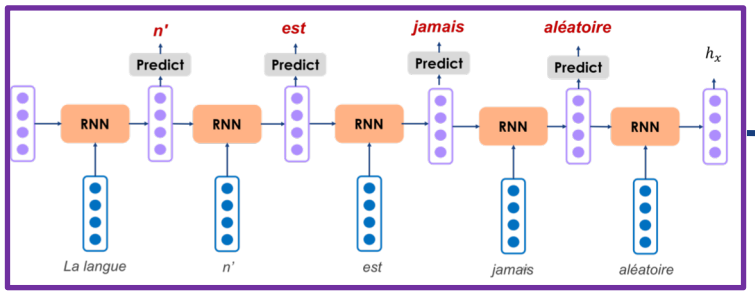
Hands-on

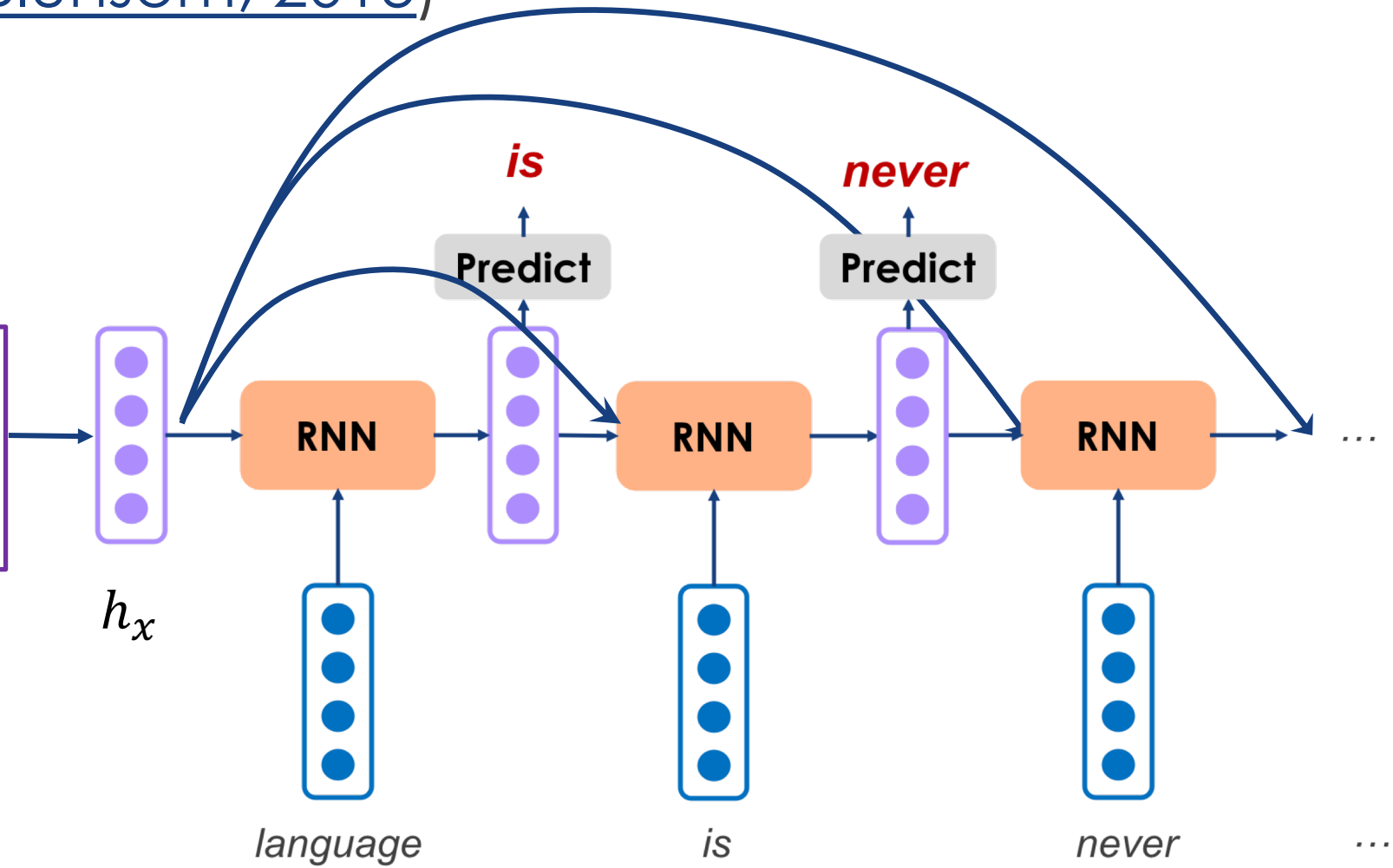
- Seq2Seq with Attention

Attention

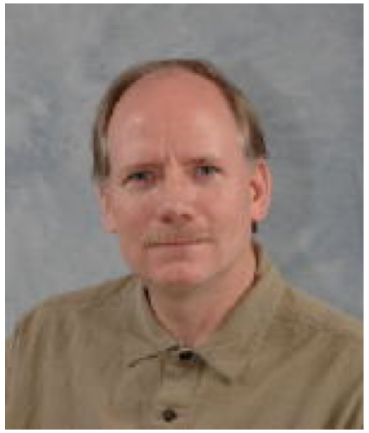
Recurrent Continuous Translation Models

- Add the encoded hidden state at every decoder time step (Kalchbrenner & Blunsom, 2013)





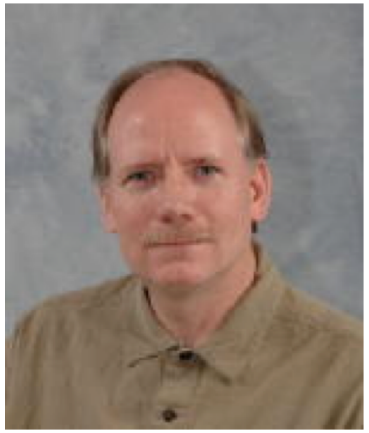
Sentence Representation



"You can't cram the meaning of a whole !@#$-ing sentence into a single %^&-ing vector!"*

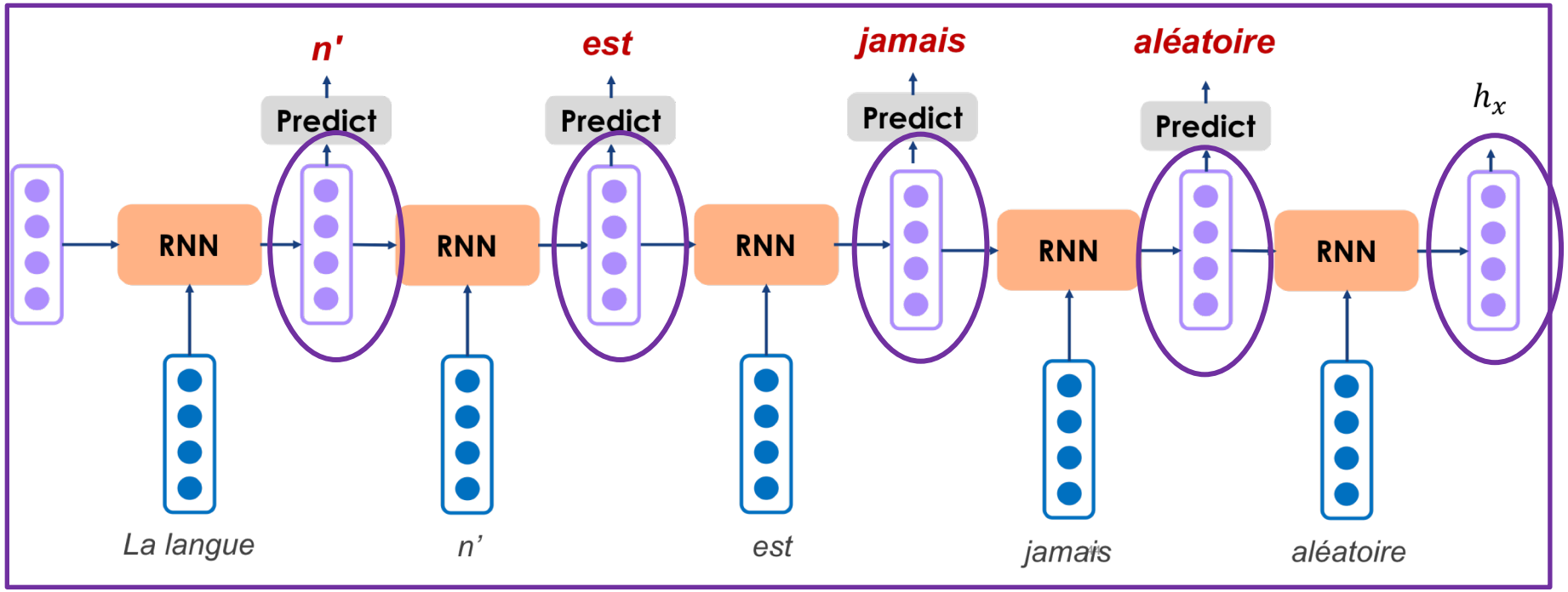
– Raymond Mooney

Sentence Representation



"You can't cram the meaning of a whole !@#$-ing sentence into a single %^&-ing vector!"*

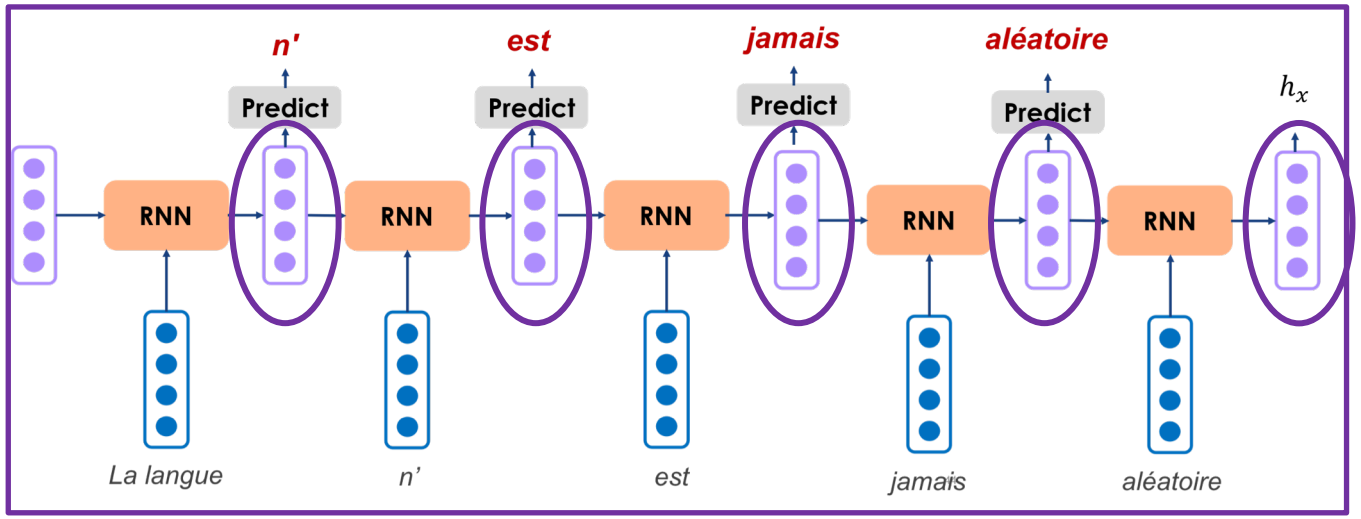
– Raymond Mooney



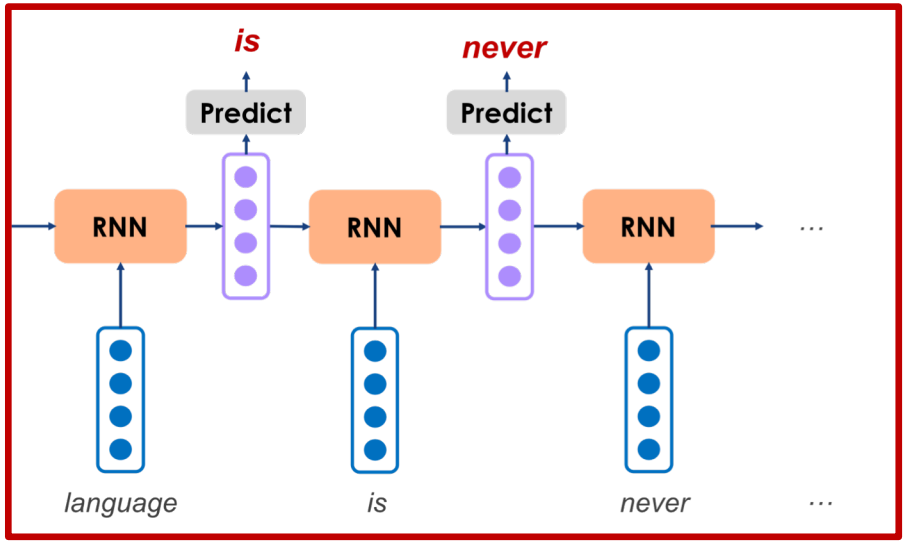
What if instead of taking just h_x, we take **all the hidden states**?

Jointly Learning to Align and Translate

- Encode each word in input into a vector
- When decoding, linearly combine encoded vector vectors weighted by "attention"
- Bahdanau et al. (2015) proposes to learn a MLP mapping between the query-key vector pairs



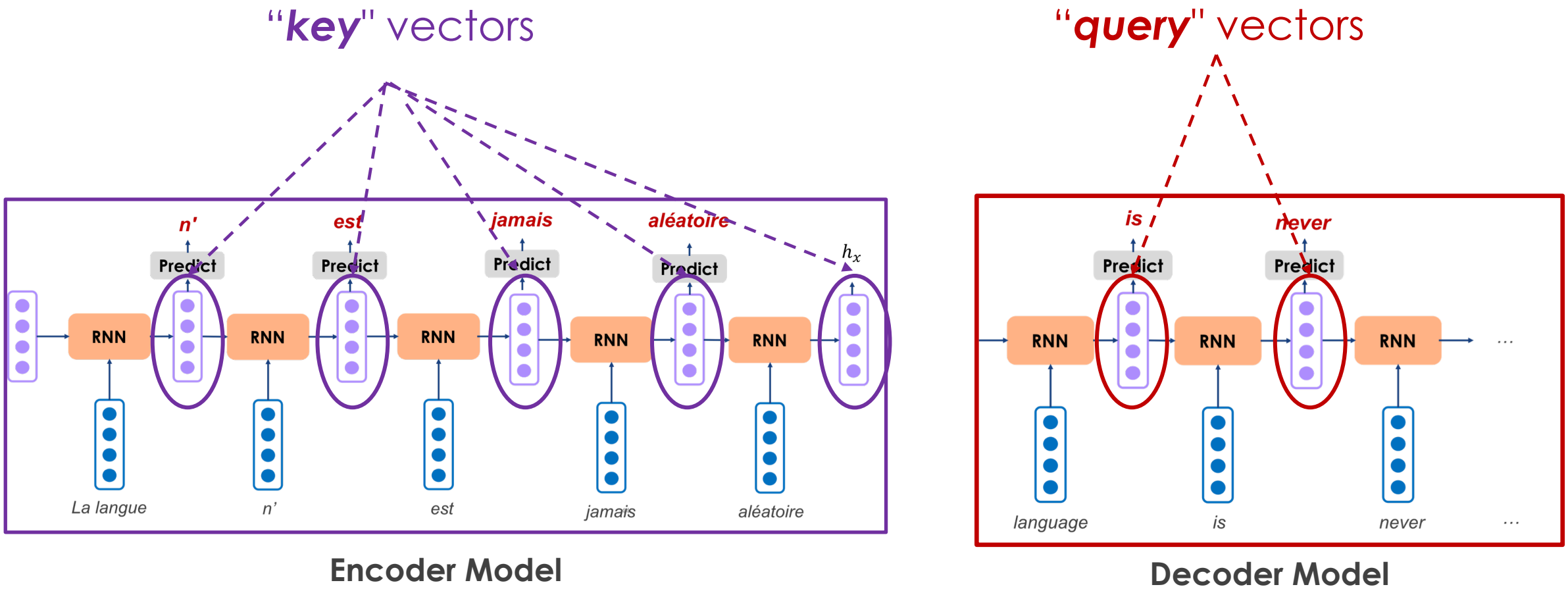
Encoder Model



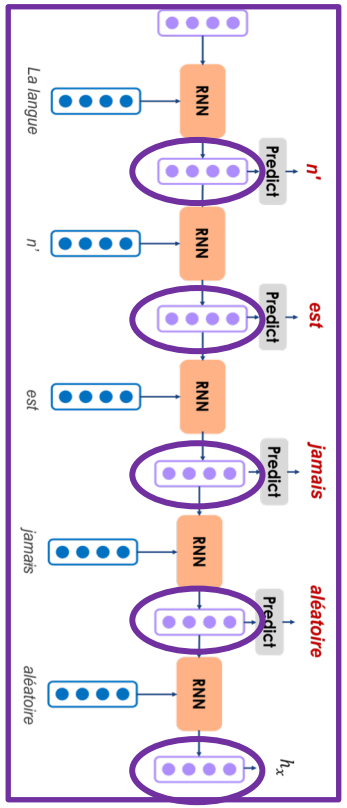
Decoder Model

Jointly Learning to Align and Translate

- Bahdanau et al. (2015) proposes to learn a MLP mapping between the query-key vector pairs



Jointly Learning to Align and Translate



Encoder Model

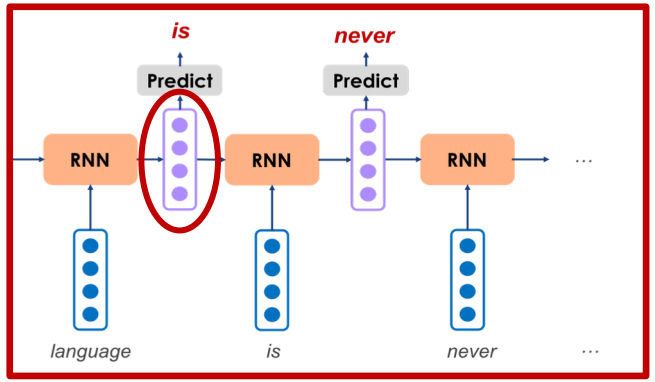
$a(q_1, k_1) = 7.5$

$a(q_1, k_2) = 1.9$

$a(q_1, k_3) = 2.1$

$a(q_1, k_4) = -3.2$

$a(q_1, k_5) = -0.5$



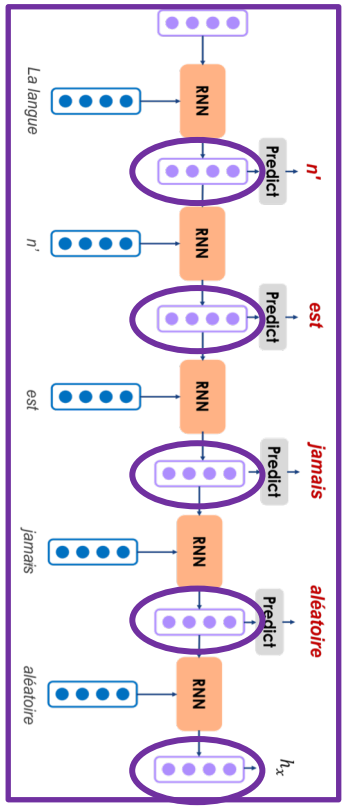
Decoder Model

Alignment score between *"key"* and *"query"*

$$a(q, k) = w_2^T(W_1[q; k])$$

Can be learned by a w_2^T vector and W_1 matrix

Jointly Learning to Align and Translate



Encoder Model

$a(q_1, k_1) = 7.5$

$a(q_1, k_2) = 1.9$

$a(q_1, k_3) = 2.1$

$a(q_1, k_4) = -3.2$

$a(q_1, k_5) = -0.5$

Softmax

0.9

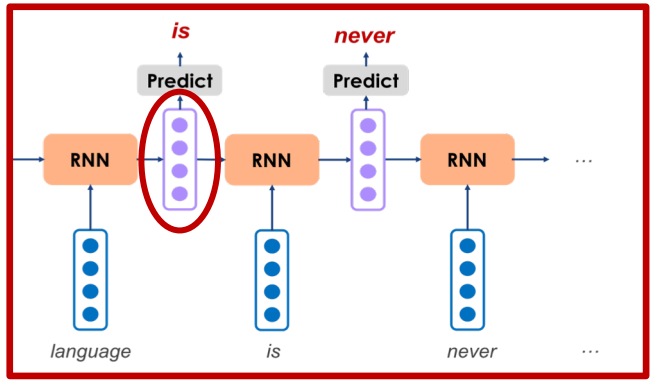
0.003

0.004

0.00005

0.0003

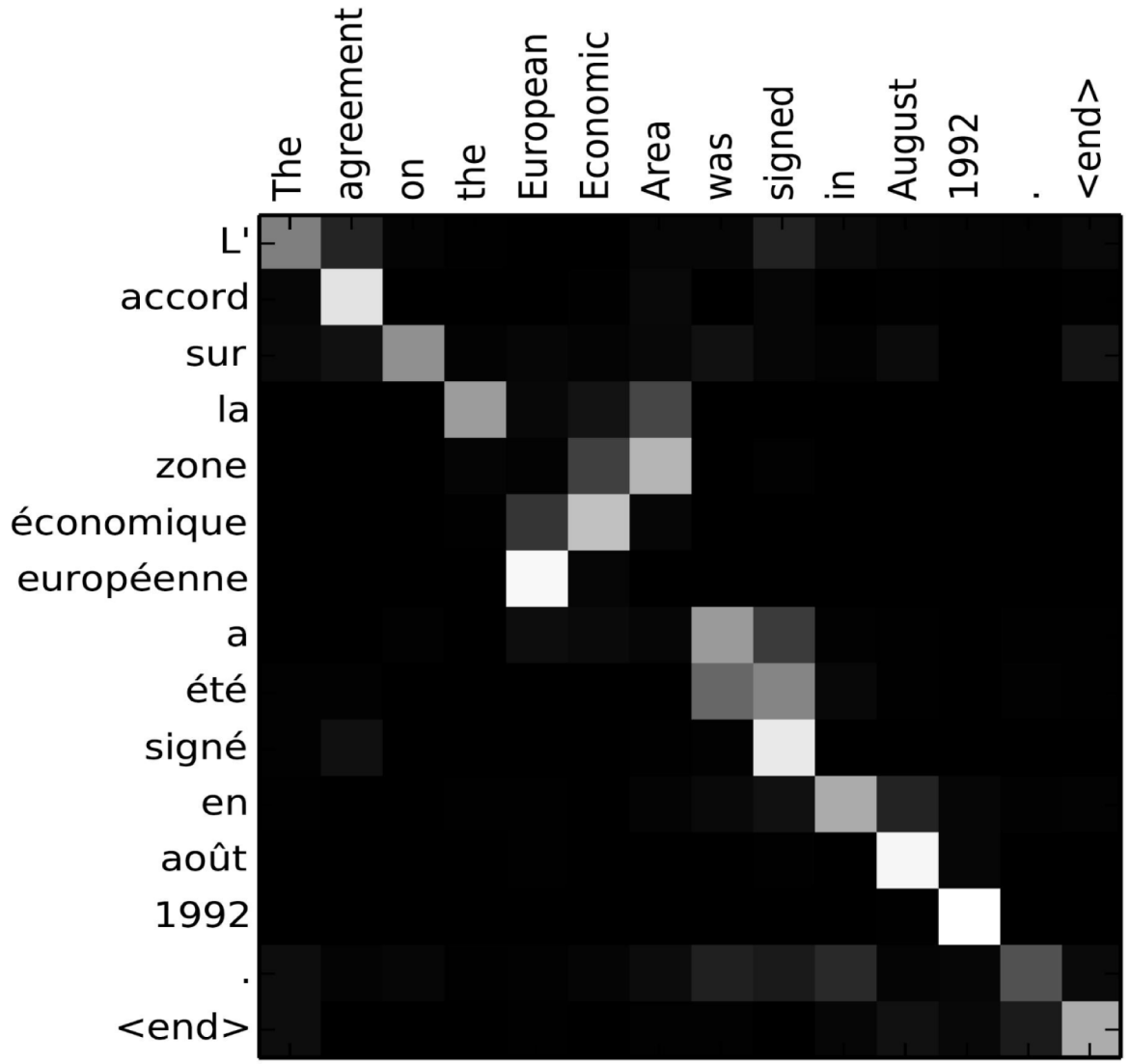
$$a(q, k) = w_2^T(W_1[q; k])$$



Decoder Model

Jointly Learning to Align and Translate

- Bahdanau et al. (2015) proposes to learn a MLP mapping between the query-key vector pairs
- Each decoder state generates a probabilistic "attention" vector that aligns the predicted word at the time step to every input word in the encoder
- As a by-product, it creates a word-alignment matrix (e.g. figure on right)



Jointly Learning to Align and Translate

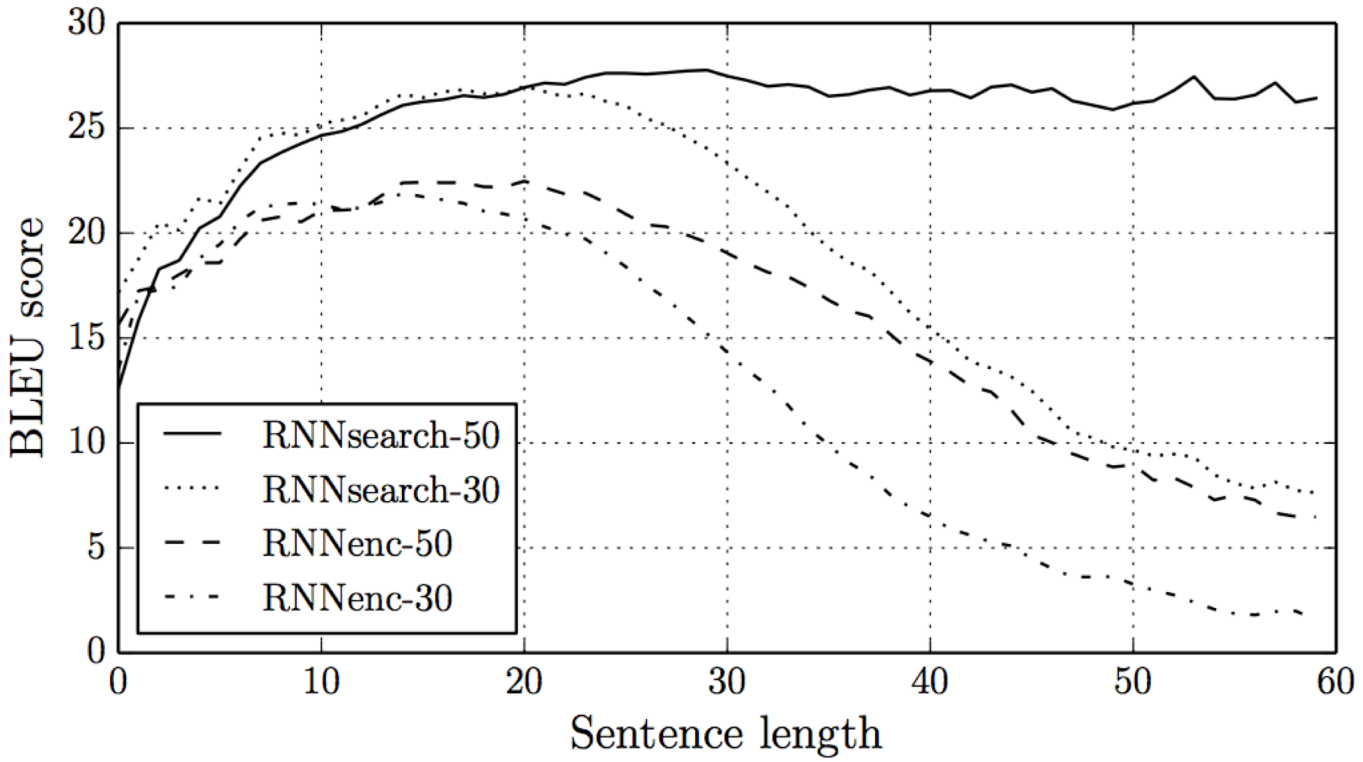


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Bahdanau et al. (2015)

Many Other Attentions

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

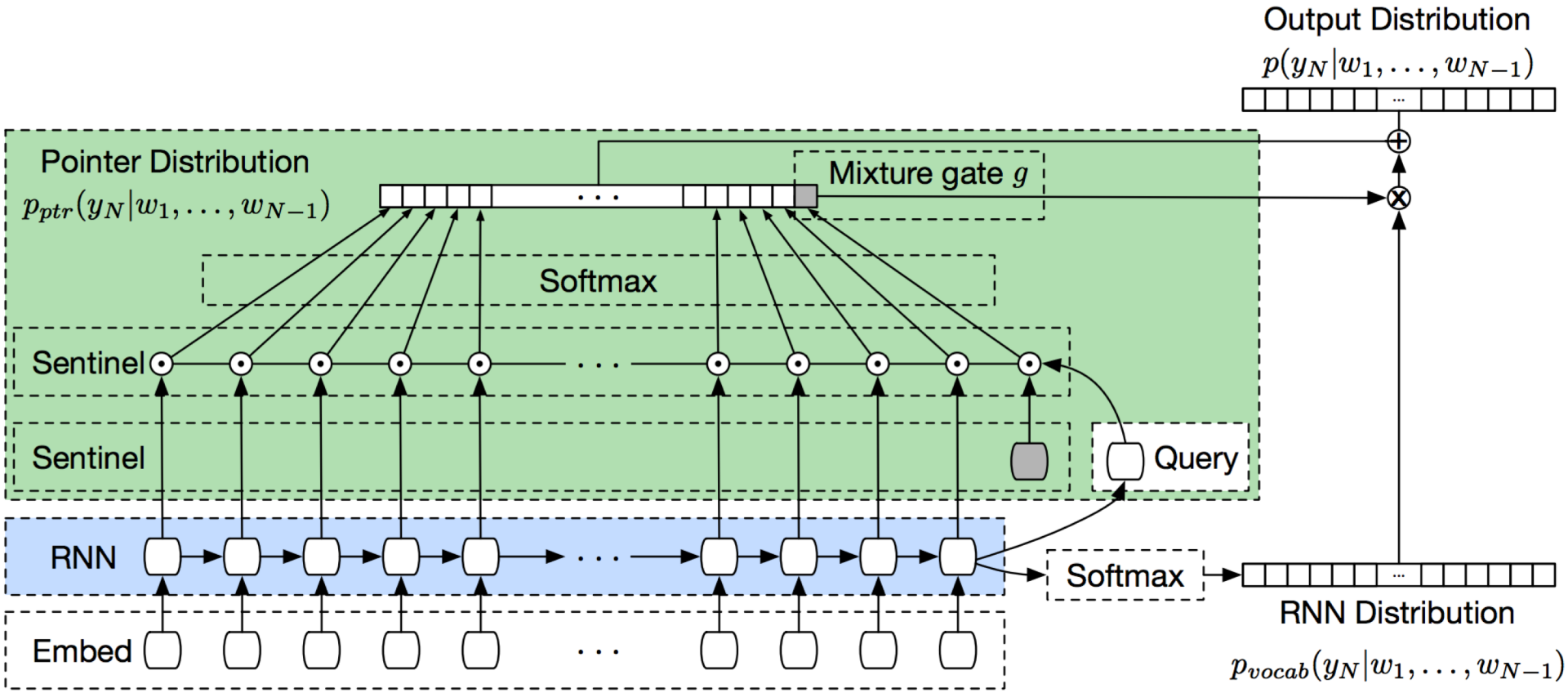
(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.

(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

(Cite from Lilian Weng's blog)

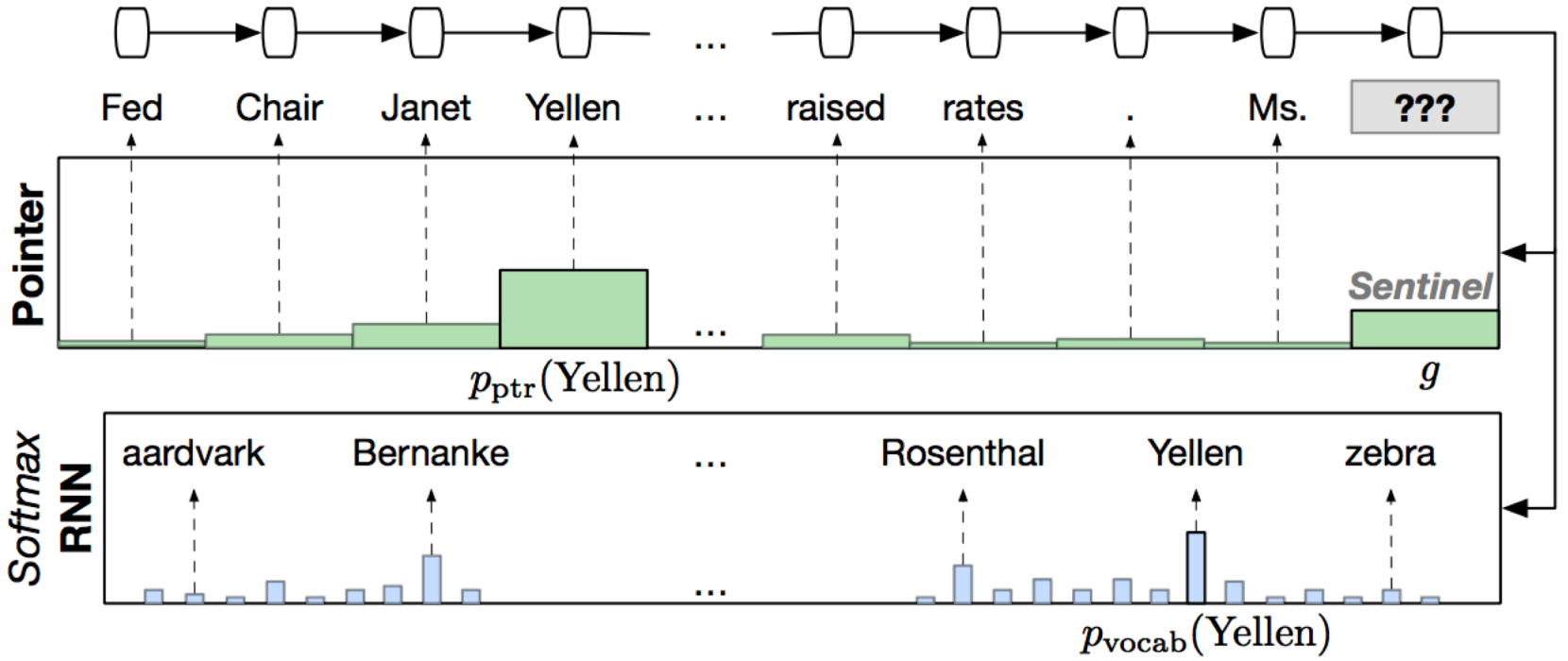
Reproducing Previous Generated Word

- Merity et al. (2016) applied a mixture model of softmax and pointers.



Reproducing Previous Generated Word

- Merity et al. (2016) applied a mixture model of softmax and pointers.



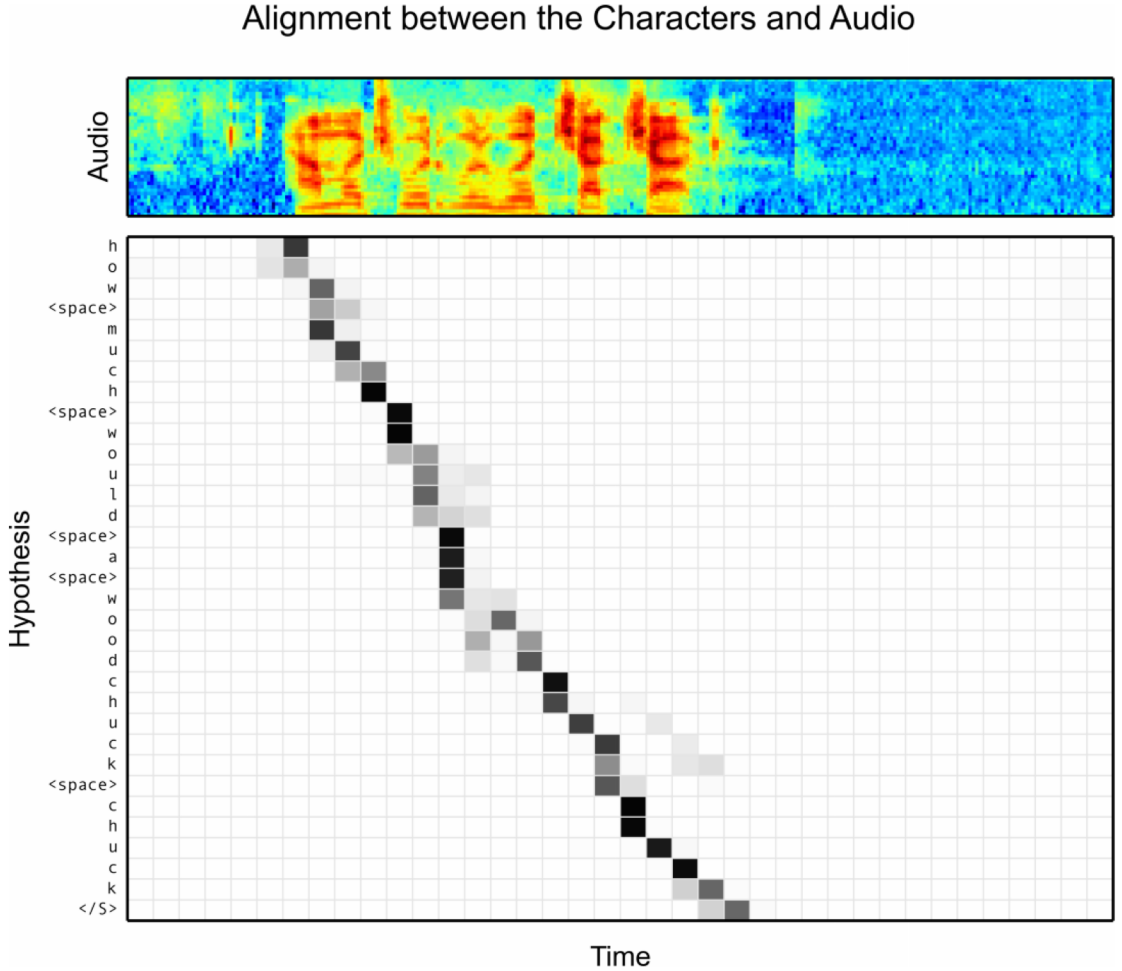
$$p(\text{Yellen}) = g\, p_{\text{vocab}}(\text{Yellen}) + (1-g)\, p_{\text{ptr}}(\text{Yellen})$$

Show, Attend and Tell

Xu et al. (2015) generate captions from images and found that attention activates for words with respect to specific regions in the image

Listen, Attend and Spell

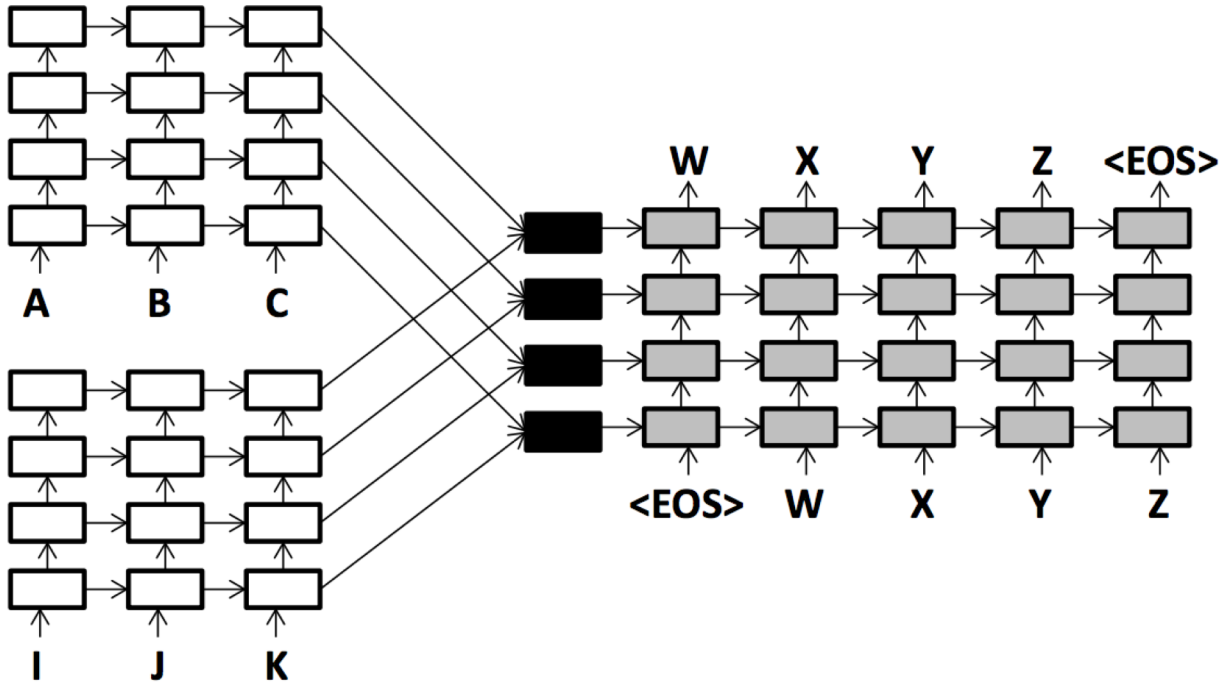
Chan et al. (2015) generate characters from audios and found that attention learns start and end of utterance

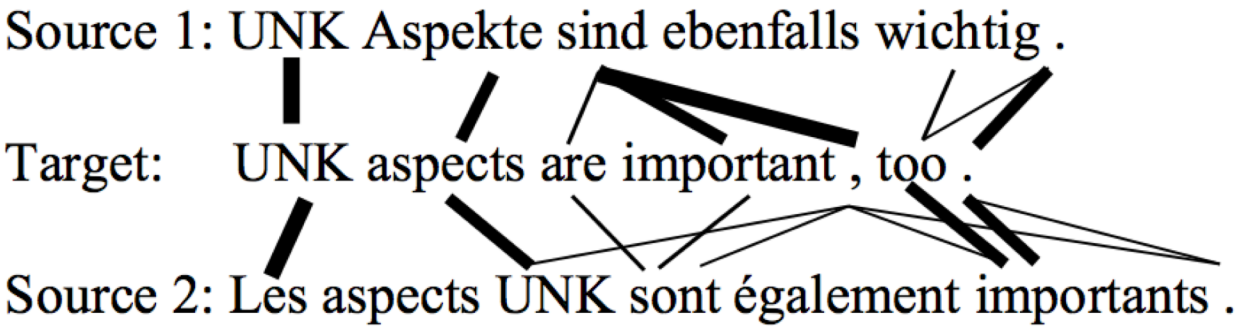


Multiple Source Attention

Zoph and Knight (2015) combines multiple encoders through the attention mechanism for NMT

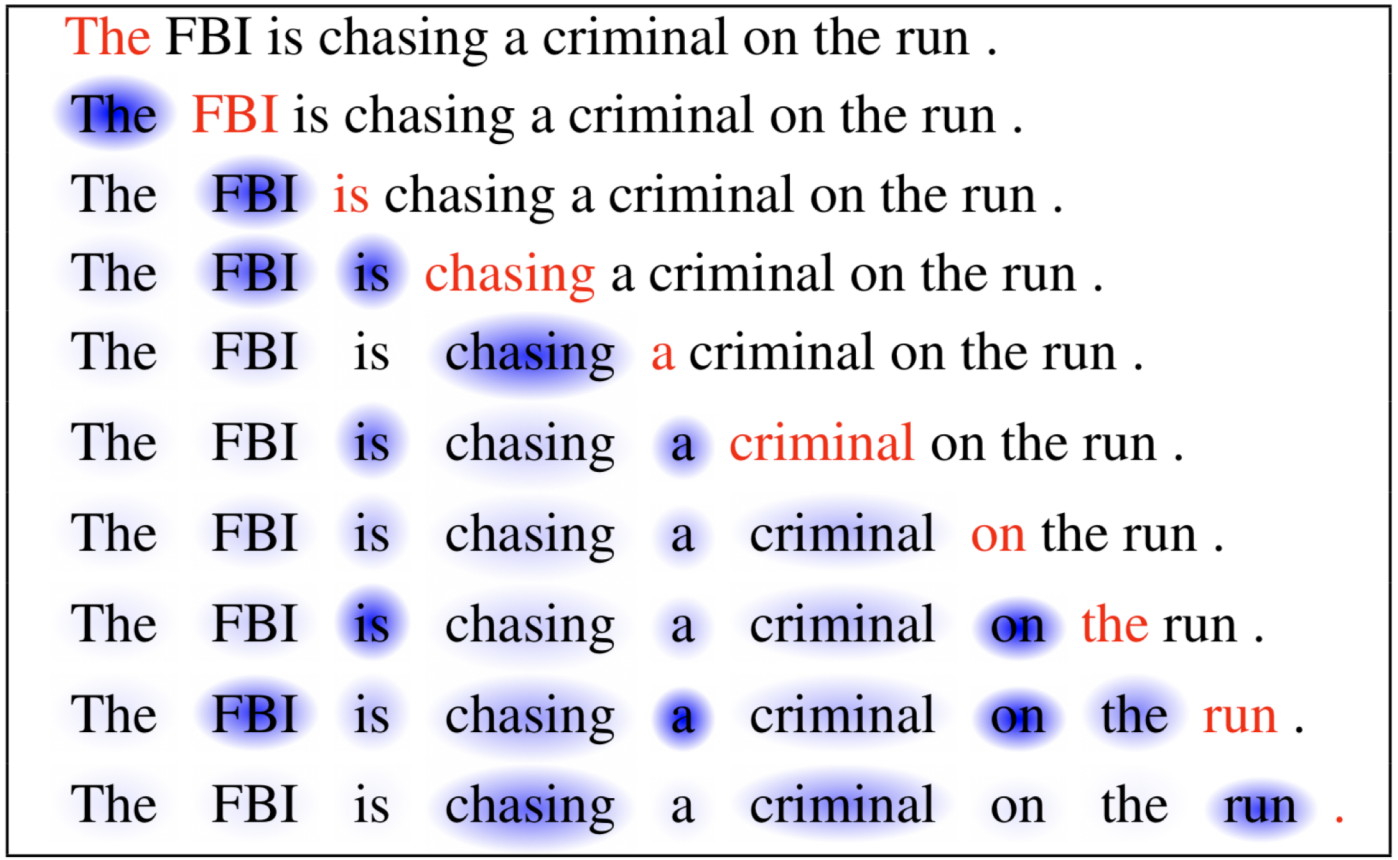
Libovicky and Helcl (2017) suggested a couple of strategies for combined attention





Self-Attention

Cheng et al. (2016) proposed **self-attention relating the current word with previous words**



Multi-headed Attention

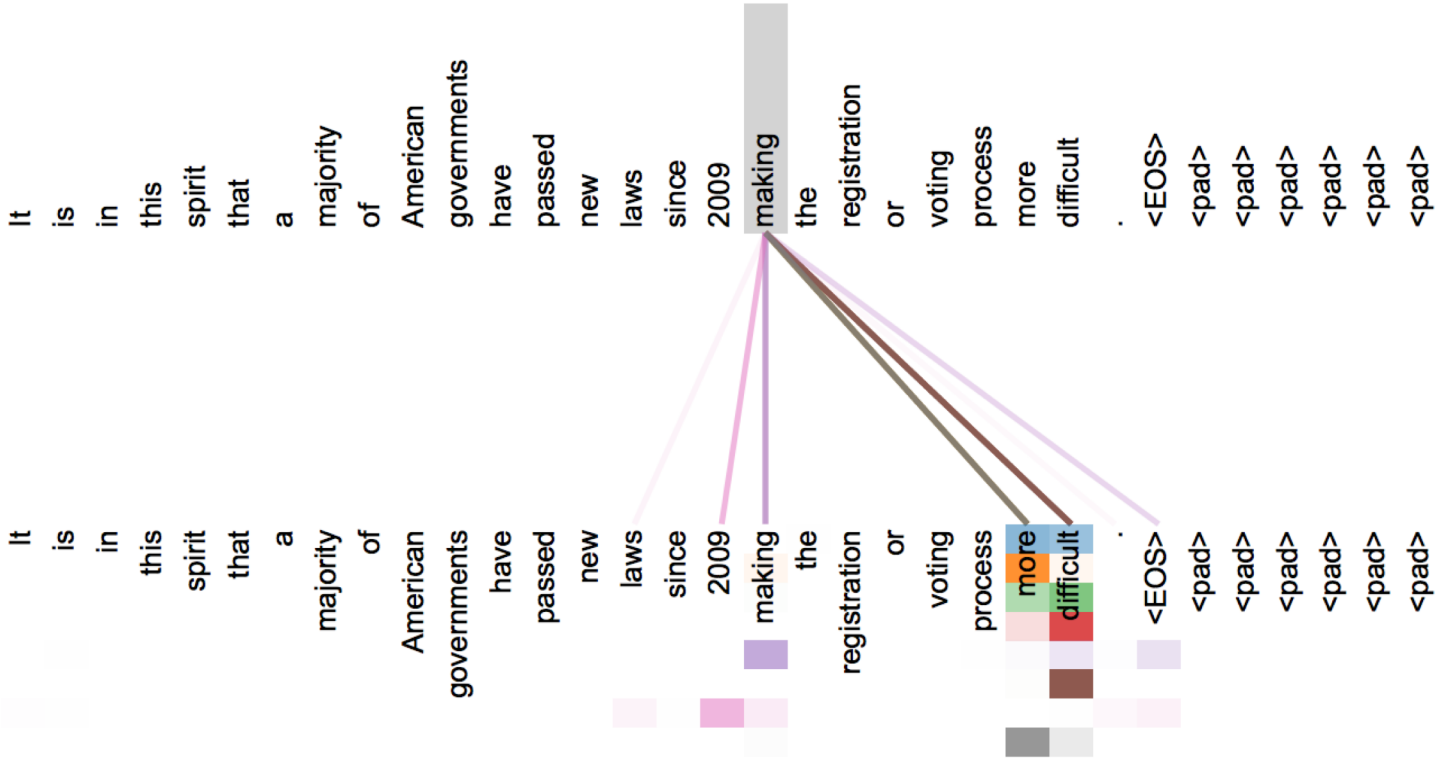
Create multiple attention "heads" to focus on different parts of the sentence.

Allamanis et al. (2016) creates different heads for "copy" vs regular convolutional attention

	Target		Attention Vectors	λ
m_1	set	$\alpha =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	0.012
		$\kappa =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	
m_2	use	$\alpha =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	0.974
		$\kappa =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	
m_3	browser	$\alpha =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	0.969
		$\kappa =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	
m_4	cache	$\alpha =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	0.583
		$\kappa =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	
m_5	END	$\alpha =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	0.066
		$\kappa =$	<s> { this . use Browser Cache = use Browser Cache ; } </s>	

Multi-headed Attention

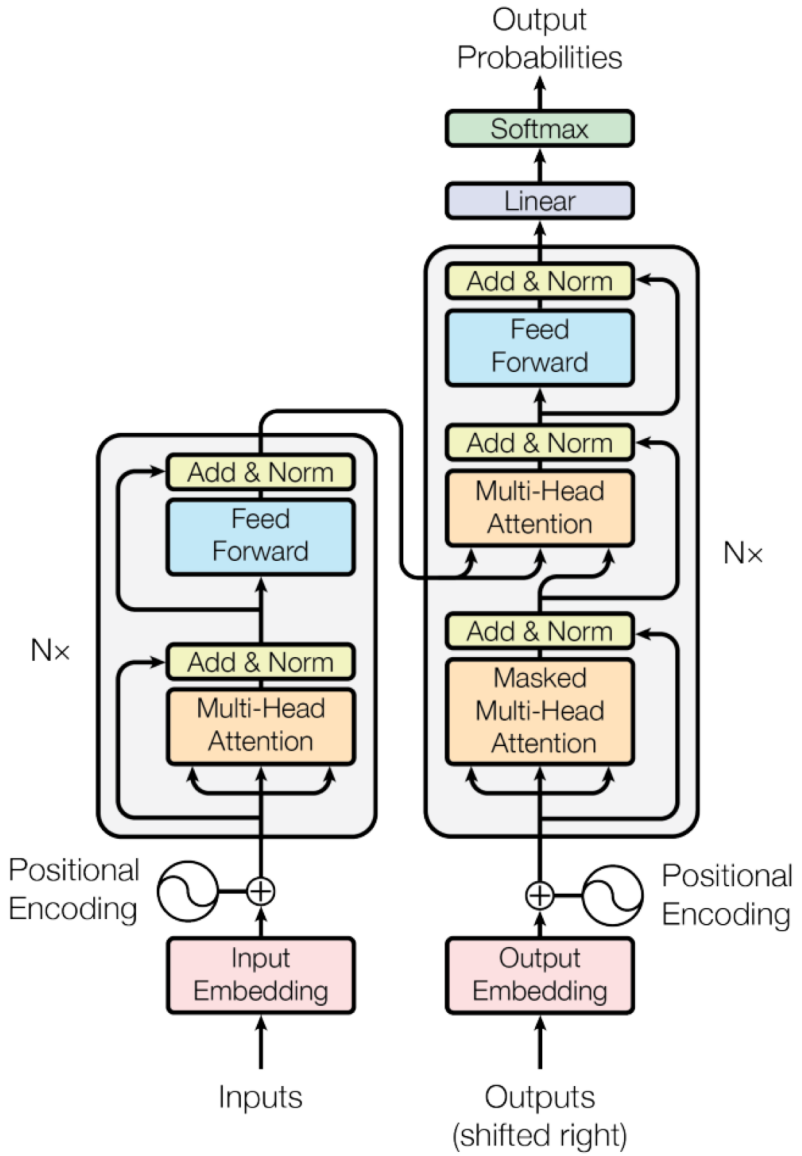
Create multiple attention "heads" to focus on different parts of the sentence.



Vaswani et al. (2017) created multiple independently learned "heads"

Attention is All You Need

- Sequence-to-Sequence model entirely based on attention, no recurrence, no convolution
- Fast because only matrix multiplication operation
- It's everywhere…



Transformer Self-Attention

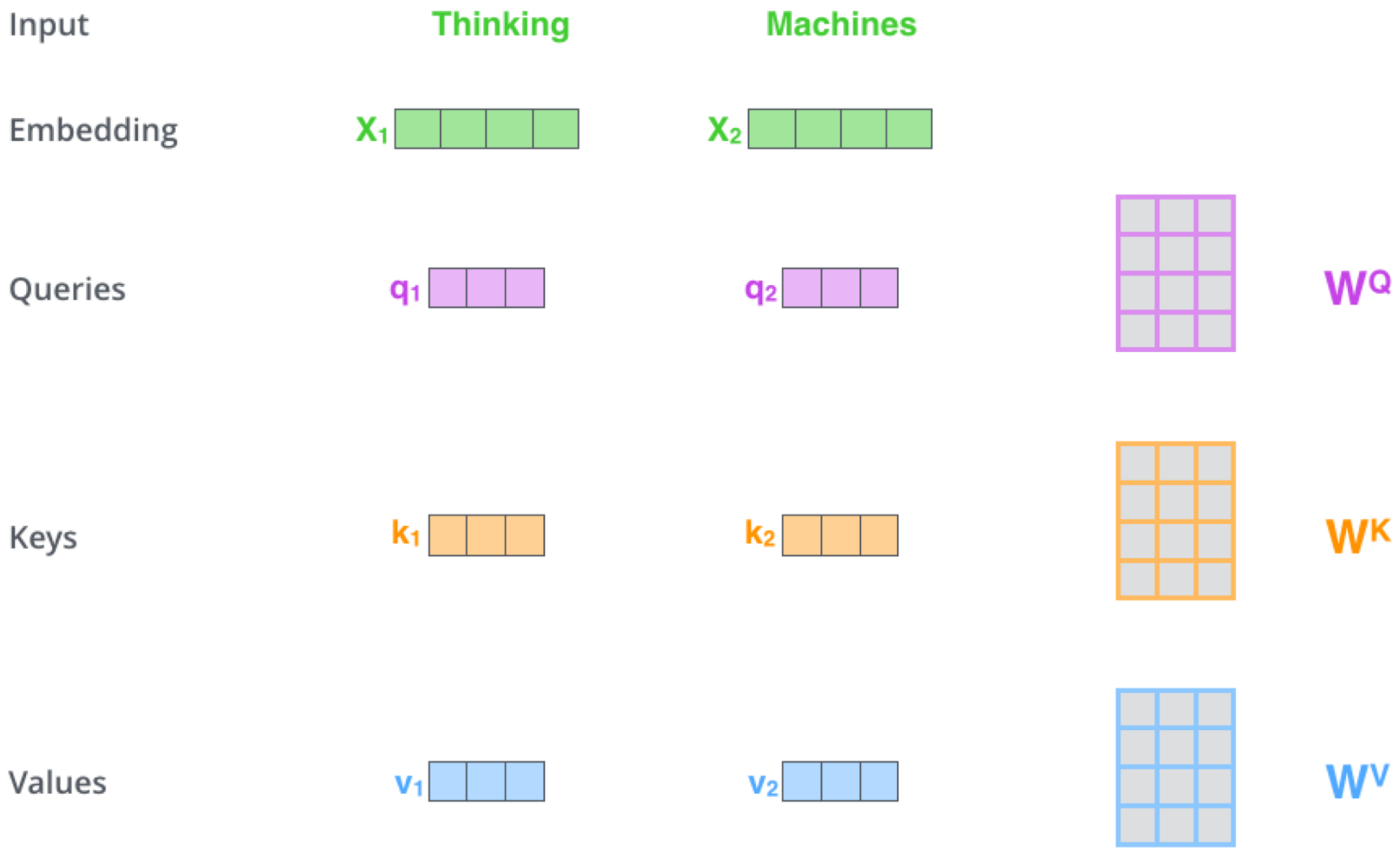


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention

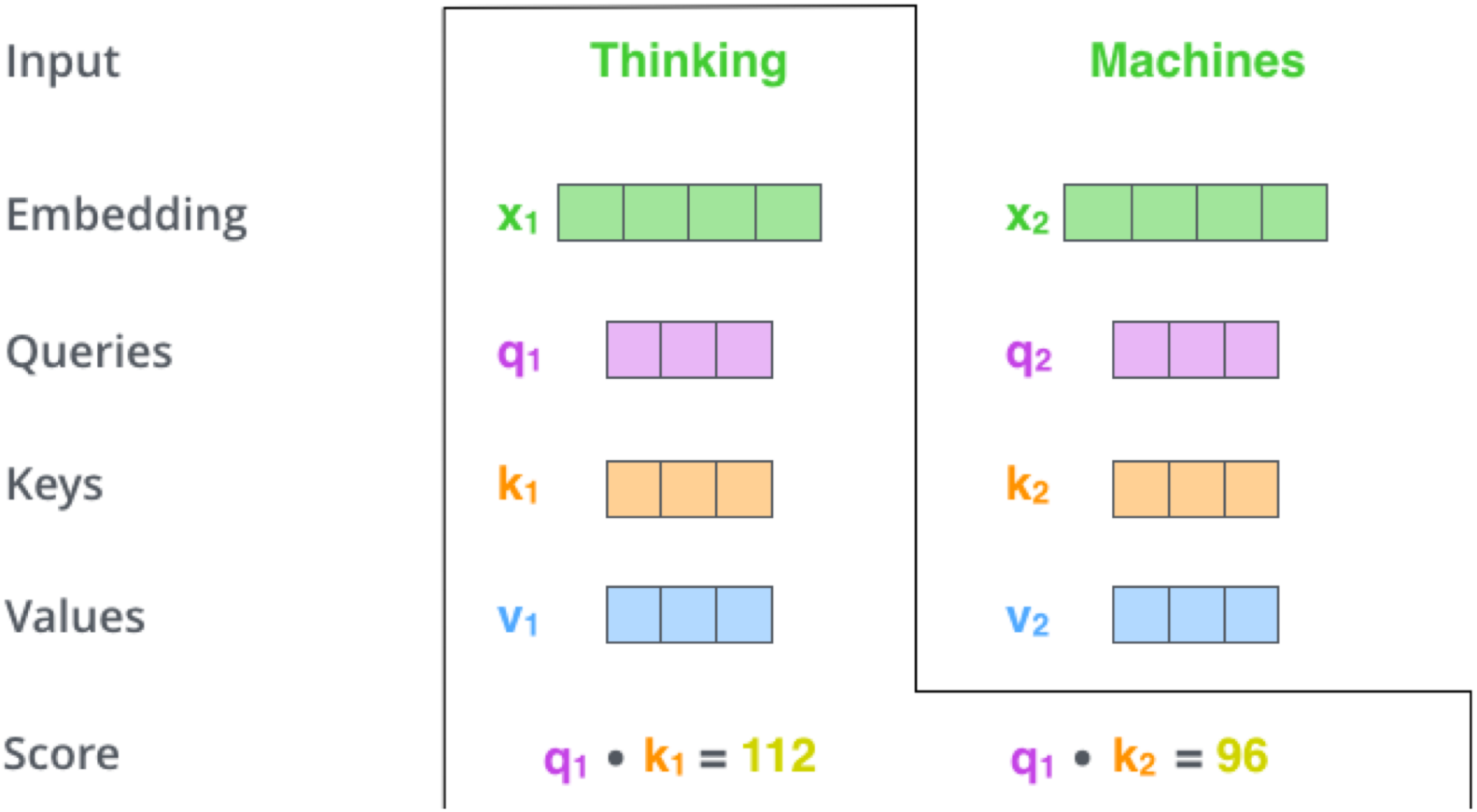


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention

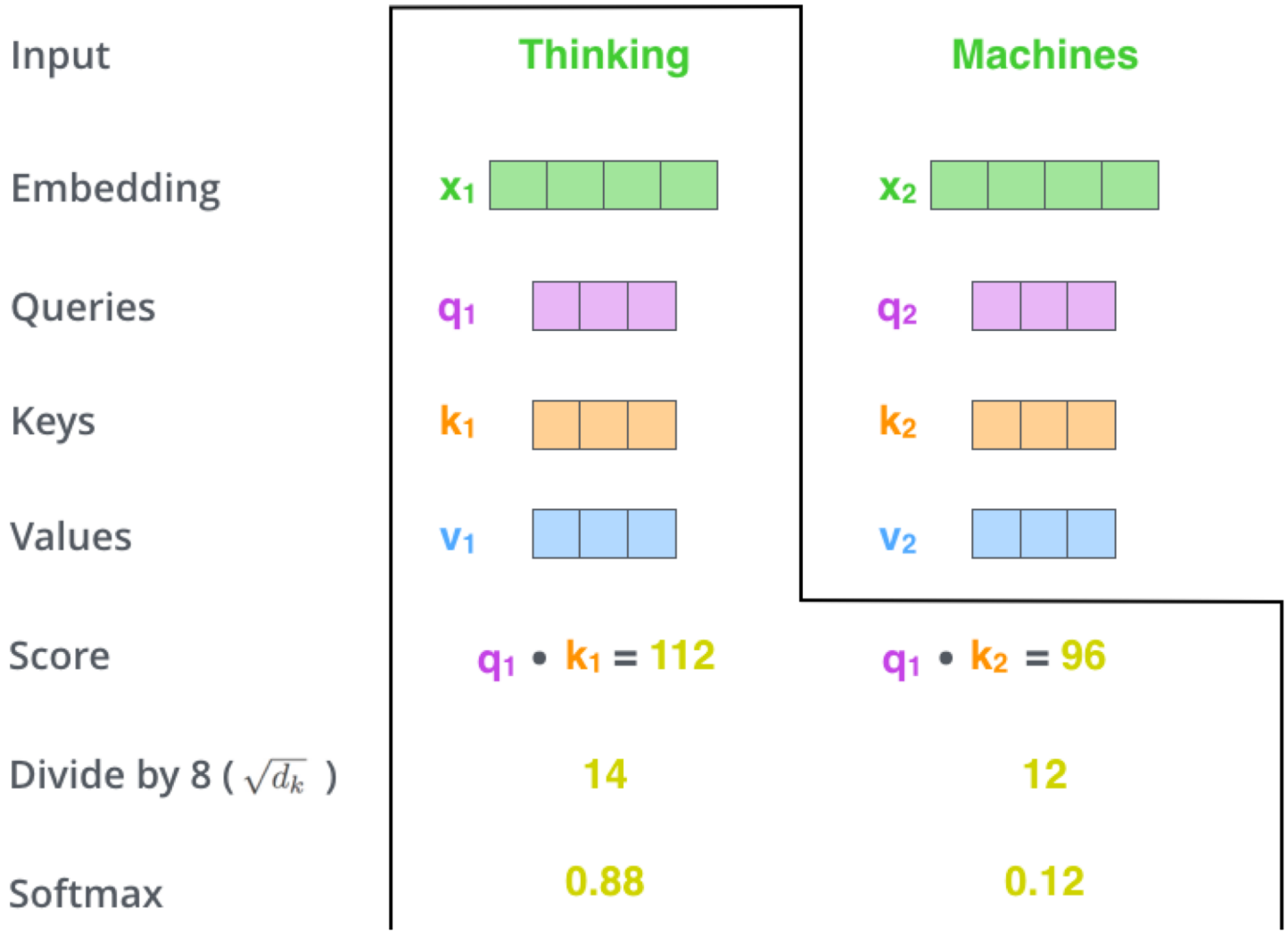


Image from http://jalammar.github.io/illustrated-transformer/

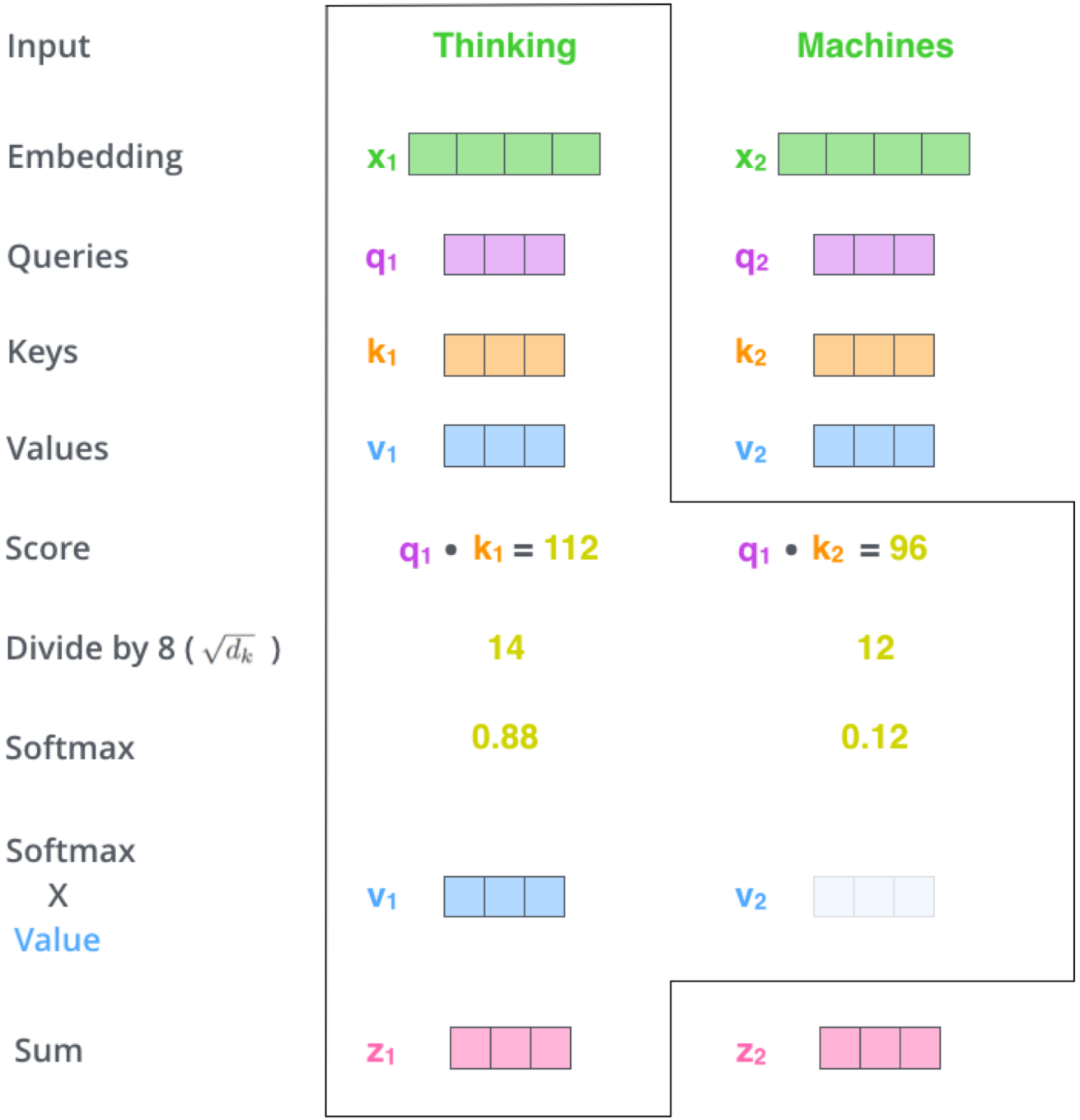
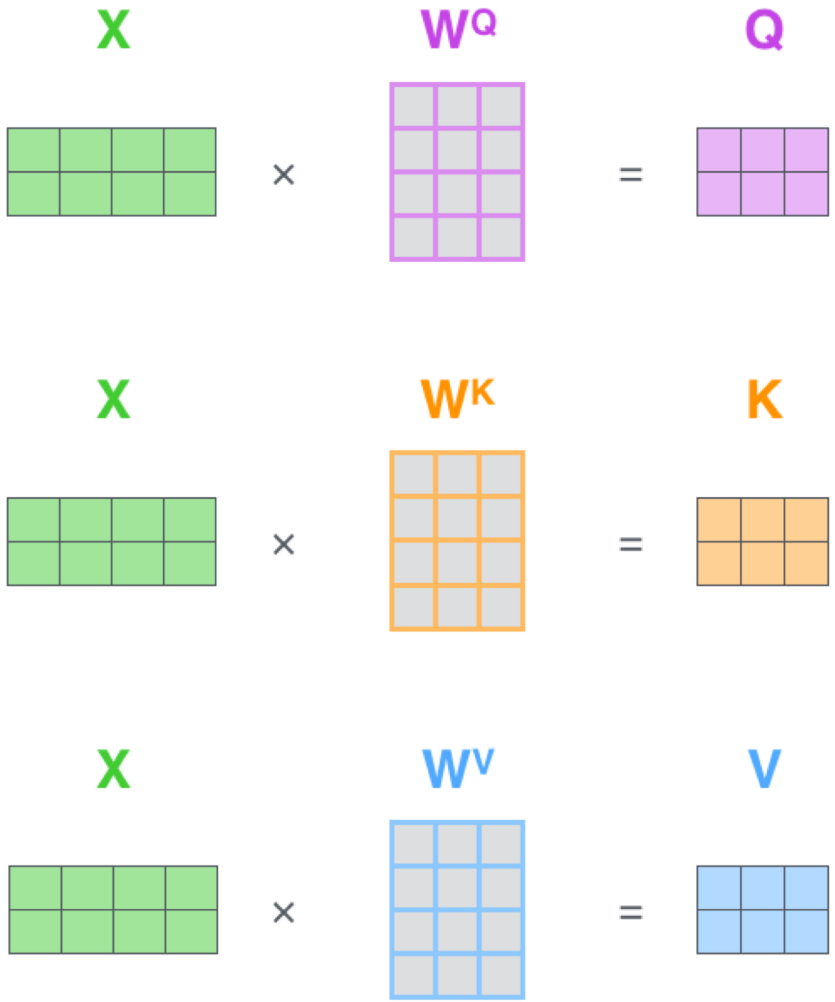


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention



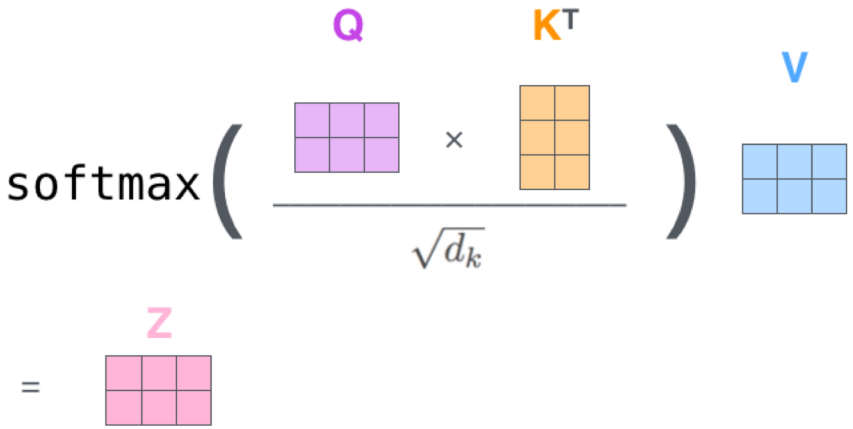


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention

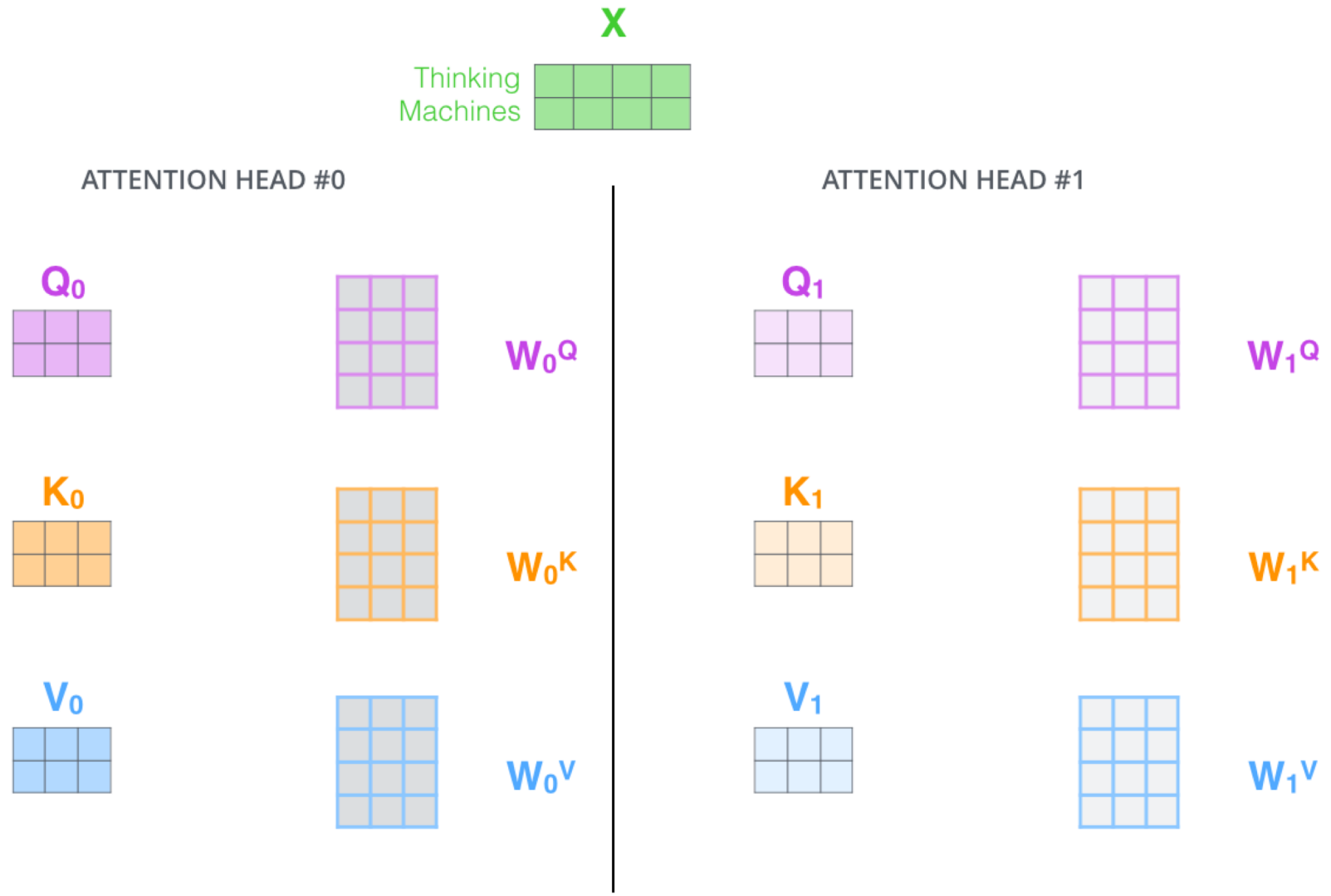


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention

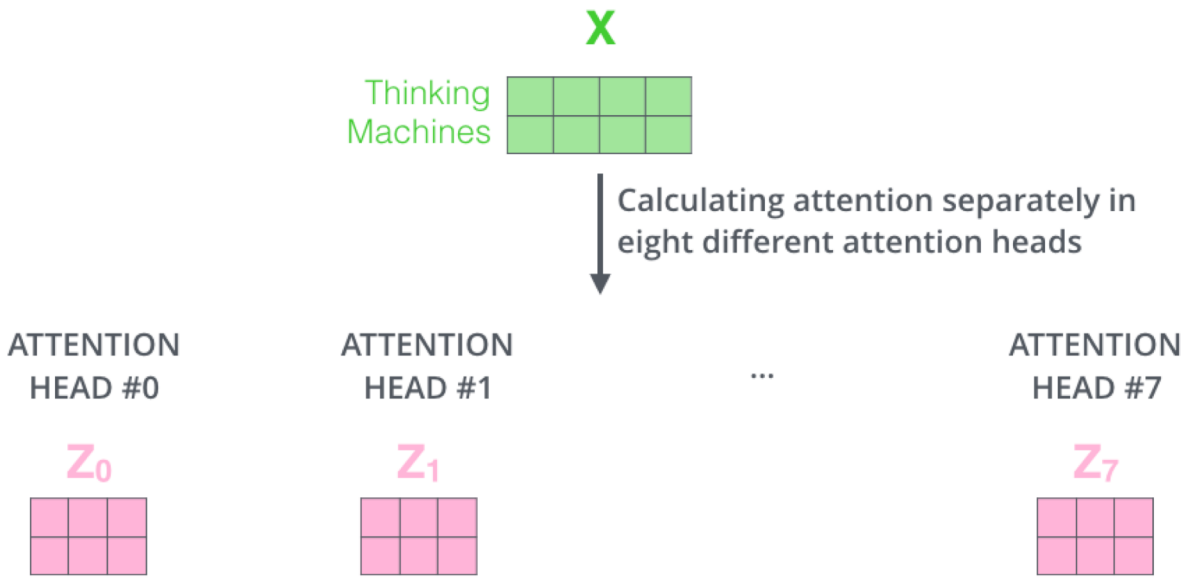
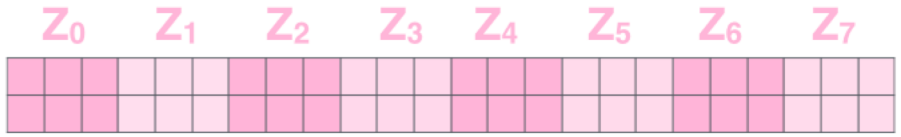


Image from http://jalammar.github.io/illustrated-transformer/

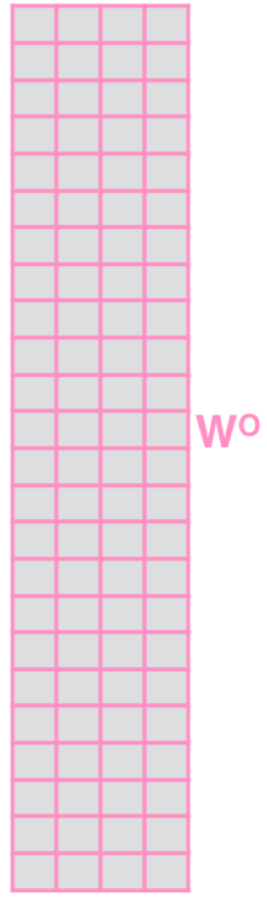
Transformer Self-Attention

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

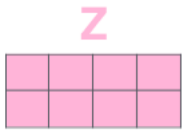


Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention

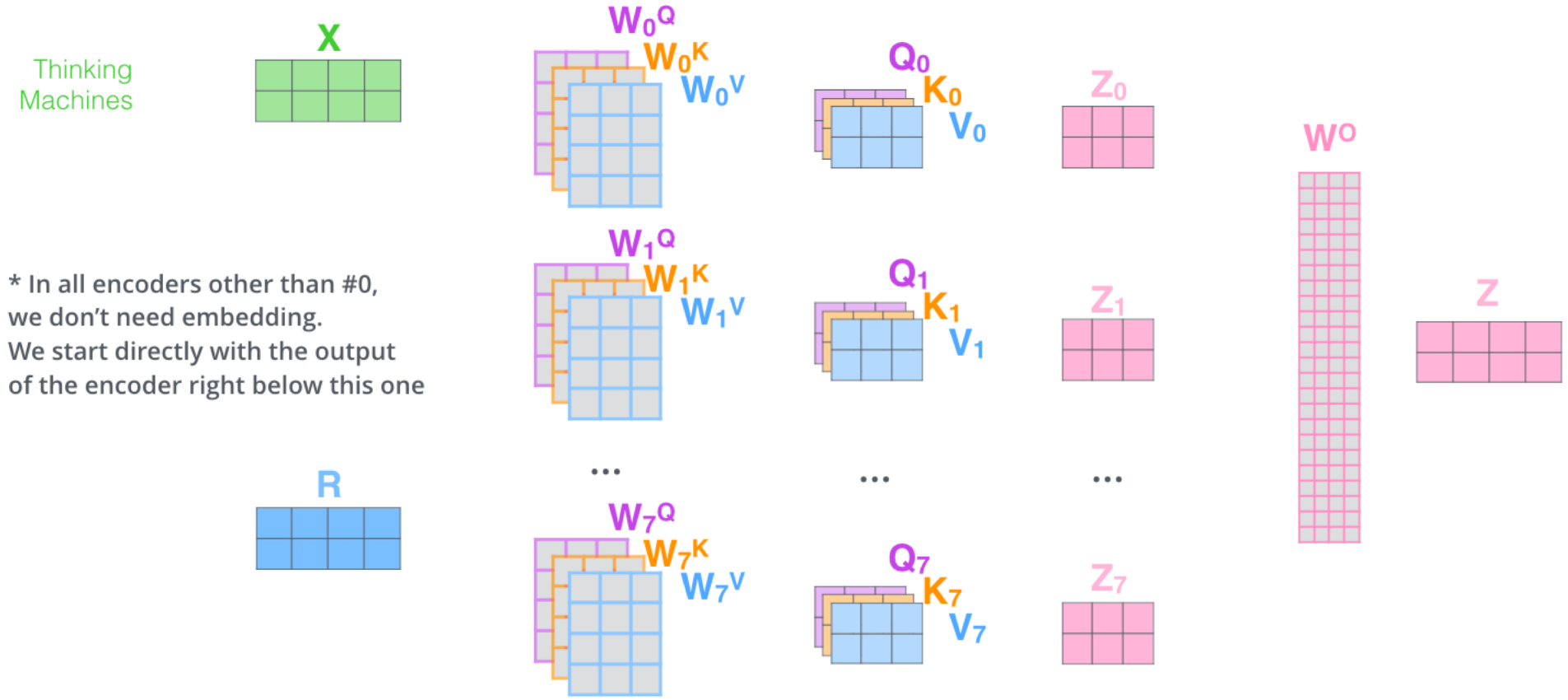
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

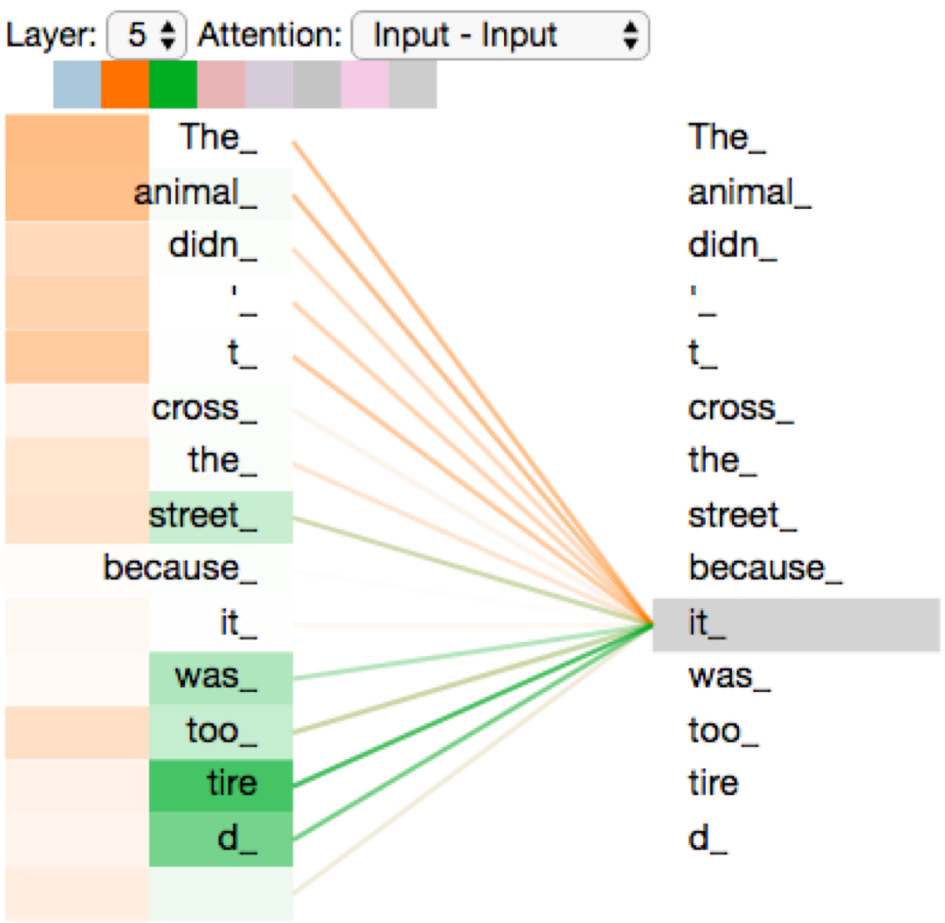
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Image from http://jalammar.github.io/illustrated-transformer/

Transformer Self-Attention



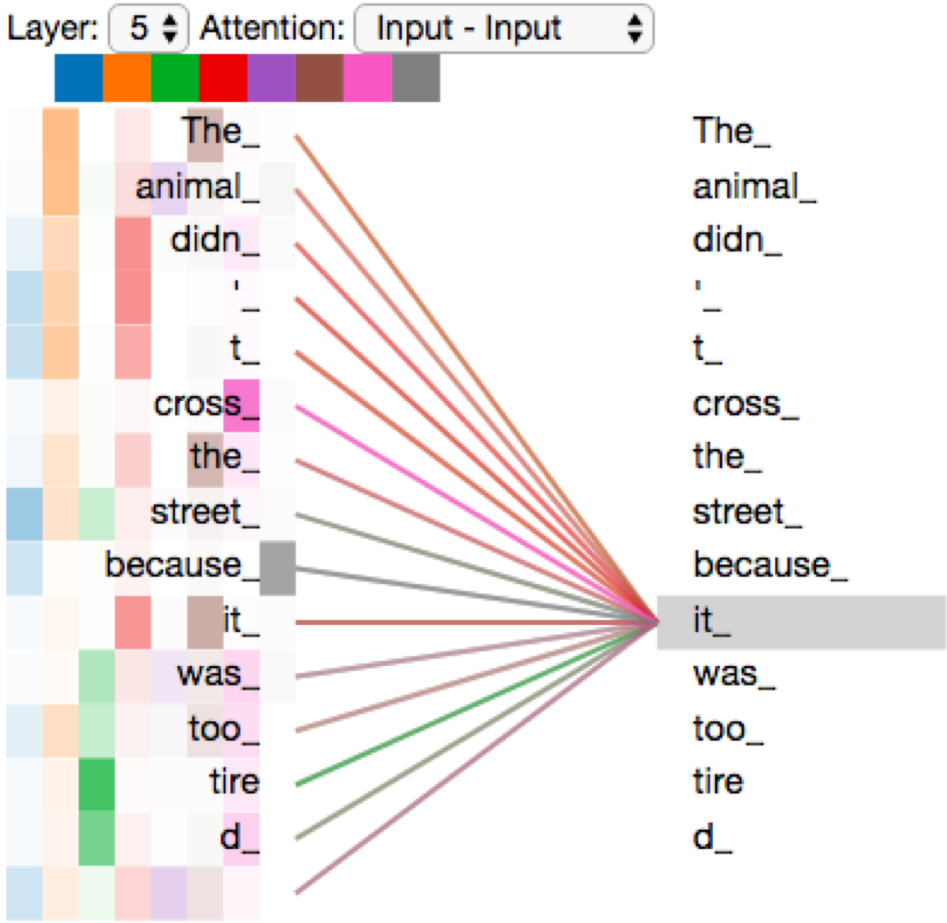


Image from http://jalammar.github.io/illustrated-transformer/

Attention Tricks

- **Self Attention:** Each layer combines words with every other word in sentence
- **Multi-headed Attention:** Multiple attention heads learnt independently
- **Normalized dot product attention:** removes bias
- **Positional Encodings:** Can distinguish positive without stepwise recurrence

Cite from Neubig (2018) CMU NN4NLP Course

Training Tricks

- **Layer Normalization:** Ensure layer outputs remains in range
- **Specialized Training Schedule:** Warm-up and cool-up scheduler adjust defaults Adam learning rates
- **Label Smoothing:** Insert some uncertainty during training to prevent overfitting

Cite from Neubig (2018) CMU NN4NLP Course

The Annotated Transformer

- Lets walk through the code

- See http://bit.ly/ANLP-AnnTransformer

Moving forward…

Lecture 8, 9 and 10…

- **Lecture 8:** Sentence Representations

- **Lecture 9 and 10:**
  - Bayesian learning and NLP
  - Syntactic and Semantic Parsing